NPS-AM-12-C9P06R03-055

EXCERPT FROM THE

PROCEEDINGS

OF THE

NINTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM WEDNESDAY SESSIONS VOLUME I

Certifying Tools for Test Reduction in Open Architecture

Valdis Berzins
Naval Postgraduate School

Published April 30, 2012

1. REPORT DATE **30 APR 2012**	2. REPORT TYPE	3. DATES COVERED **00-00-2012 to 00-00-2012**
4. TITLE AND SUBTITLE **Certifying Tools for Test Reduction in Open Architecture**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Monterey,CA,93943**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		

14. ABSTRACT

In this paper, we describe a method for evaluating tools that can be used to guide decisions about how much retesting is needed and to check conditions under which testing of unmodified components can be reduced or avoided. The approach uses a combination of dependency analysis applied to source code and automated testing applied to executable component implementations. Dependability of such tools is a key concern in this context which our ongoing research addresses. We also discuss other applications of software dependency analysis, such as risk-based testing, and discuss applications of dependency analysis to improve acquisition processes in the context of open architectures (OA). The Navy?s OA framework is intended to promote reuse, improve system flexibility, and reduce costs. In this paper, we apply open architecture principles to reduce testing effort and costs in cases where the requirements and code for a subsystem have not been changed but the component will be used together with new or modified components that may include a new version of the operating system. This situation is common in the Navy due to technology advancement upgrades and accounts for a substantial fraction of the testing cost. Applying traditional U.S. Navy weapon and combat system test and evaluation (T&E) practices, which currently include manual retesting after each system modification, to future OA systems will nullify many of the benefits that OA brings to the table, such as system scalability, rapid configuration changes, and effective component reuse. Combining (1) Naval Postgraduate School (NPS) research on dependency analysis focused on determining when it is safe not to retest a component with (2) automated software testing should enable these benefits and keep resource requirements at feasible levels.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**37**	

Preface & Acknowledgements

Welcome to our Ninth Annual Acquisition Research Symposium! This event is the highlight of the year for the Acquisition Research Program (ARP) here at the Naval Postgraduate School (NPS) because it showcases the findings of recently completed research projects—and that research activity has been prolific! Since the ARP's founding in 2003, over 800 original research reports have been added to the acquisition body of knowledge. We continue to add to that library, located online at www.acquisitionresearch.net, at a rate of roughly 140 reports per year. This activity has engaged researchers at over 60 universities and other institutions, greatly enhancing the diversity of thought brought to bear on the business activities of the DoD.

We generate this level of activity in three ways. First, we solicit research topics from academia and other institutions through an annual Broad Agency Announcement, sponsored by the USD(AT&L). Second, we issue an annual internal call for proposals to seek NPS faculty research supporting the interests of our program sponsors. Finally, we serve as a "broker" to market specific research topics identified by our sponsors to NPS graduate students. This three-pronged approach provides for a rich and broad diversity of scholarly rigor mixed with a good blend of practitioner experience in the field of acquisition. We are grateful to those of you who have contributed to our research program in the past and hope this symposium will spark even more participation.

We encourage you to be active participants at the symposium. Indeed, active participation has been the hallmark of previous symposia. We purposely limit attendance to 350 people to encourage just that. In addition, this forum is unique in its effort to bring scholars and practitioners together around acquisition research that is both relevant in application and rigorous in method. Seldom will you get the opportunity to interact with so many top DoD acquisition officials and acquisition researchers. We encourage dialogue both in the formal panel sessions and in the many opportunities we make available at meals, breaks, and the day-ending socials. Many of our researchers use these occasions to establish new teaming arrangements for future research work. In the words of one senior government official, "I would not miss this symposium for the world as it is the best forum I've found for catching up on acquisition issues and learning from the great presenters."

We expect affordability to be a major focus at this year's event. It is a central tenet of the DoD's Better Buying Power initiatives, and budget projections indicate it will continue to be important as the nation works its way out of the recession. This suggests that research with a focus on affordability will be of great interest to the DoD leadership in the year to come. Whether you're a practitioner or scholar, we invite you to participate in that research.

We gratefully acknowledge the ongoing support and leadership of our sponsors, whose foresight and vision have assured the continuing success of the ARP:

- Office of the Under Secretary of Defense (Acquisition, Technology, & Logistics)
- Director, Acquisition Career Management, ASN (RD&A)
- Program Executive Officer, SHIPS
- Commander, Naval Sea Systems Command
- Program Executive Officer, Integrated Warfare Systems
- Army Contracting Command, U.S. Army Materiel Command
- Office of the Assistant Secretary of the Air Force (Acquisition)

- Office of the Assistant Secretary of the Army (Acquisition, Logistics, & Technology)
- Deputy Director, Acquisition Career Management, U.S. Army
- Office of Procurement and Assistance Management Headquarters, Department of Energy
- Director, Defense Security Cooperation Agency
- Deputy Assistant Secretary of the Navy, Research, Development, Test & Evaluation
- Program Executive Officer, Tactical Aircraft
- Director, Office of Small Business Programs, Department of the Navy
- Director, Office of Acquisition Resources and Analysis (ARA)
- Deputy Assistant Secretary of the Navy, Acquisition & Procurement
- Director of Open Architecture, DASN (RDT&E)
- Program Executive Officer, Littoral Combat Ships

We also thank the Naval Postgraduate School Foundation and acknowledge its generous contributions in support of this symposium.

James B. Greene Jr.	Keith F. Snider, PhD
Rear Admiral, U.S. Navy (Ret.)	Associate Professor

Panel 6. Considerations in Acquiring Open Architecture Software Systems

Wednesday, May 16, 2012	
1:45 p.m. – 3:15 p.m.	**Chair: Captain Joseph J. Beel**, USN, Commanding Officer, Space and Naval Warfare Systems Center Pacific ***A Framework for Reuse in the DoN*** Randy Mactal, *Space and Naval Warfare Systems Center Pacific* Lynne Spruill, *APEO Engineering Support* ***Addressing Challenges in the Acquisition of Secure Software Systems With Open Architectures*** Walt Scacchi and Thomas Alspaugh *University California, Irvine* ***Certifying Tools for Test Reduction in Open Architecture*** Valdis Berzins, *Naval Postgraduate School*

Joseph J. Beel—Captain Joe Beel was commissioned from the U.S. Naval Academy in 1985, earning a Bachelor of Science degree in mechanical engineering. He was designated a Naval Aviator in September 1986. He completed Fleet Replacement Pilot training with HSL-31 in May 1987 and joined the Sea Snakes of HSL-33, flying the SH-2F Sea Sprite until December 1989. He deployed in the USS *Kirk* (FF1067), the USS *Knox* (FF 1052), the USS *Francis Hammond* (FF1067), and the USS *Sterrett* (CG 31), including service in Operation Earnest Will.

He attended the Naval Postgraduate School in Monterey, CA, from 1990 until 1992, earning a Master of Science (with distinction) in operations research. He taught in the U.S. Naval Academy Mathematics Department from May 1992 until May 1995 and served as the Fifth Company Officer from August 1993 until May 1995. He also served as an advanced seamanship and navigation instructor and was designated a craftmaster/yard patrol craft officer-in-charge afloat.

Captain Beel completed Fleet Replacement Pilot training with HSL-41 in February 1996 and joined the Battle Cats of HSL-43, flying the SH-60B Sea Hawk until 1998. He deployed in the USS *Princeton* (CG 59).

From June 1998 until August 1999, Captain Beel served as the training and education program analyst in the Assessment Division (N81), Office of the Chief of Naval Operations. He served in a Federal Executive Fellowship at the RAND Corporation in Santa Monica, CA, from August 1999 to August 2000. From August 2000 until September 2002, he served in the USS *John C. Stennis* (CVN 74), including service in Operations Noble Eagle and Enduring Freedom. He served as officer-in-charge of Navy Warfare Development Command, Detachment San Diego, from October 2002 until August 2003. He served as commanding officer and executive officer, Naval Air Technical Data and Engineering Service Command (NATEC), from September 2003 until September 2006.

Most recently, Captain Beel served four years in the Program Executive Office (PEO), Command, Control, Communication, Computers, and Intelligence (C4I); as PEO chief of staff and deputy for Operations from October 2006 to June 2008; and as deputy program manager of the Navy Tactical Networks Program Office from June 2008 to August 2010.

Captain Beel is a member of the Defense Acquisition Corps and is Level III certified in Program Management, Life Cycle Logistics and Production, and Quality and Manufacturing. He is a certified

Lean Six Sigma Black Belt. He led a continuous process improvement project that was awarded a California Council of Excellence California Team Excellence bronze award and was selected to compete for the American Society of Quality's International Team Excellence Award at the 2011 World Conference on Quality and Improvement.

Captain Beel's awards include the Meritorious Service Medal (three awards), Air Medal (individual award), Navy Commendation Medal (five awards), Navy Achievement Medal, and various unit, campaign, and service awards. He has also received the Sikorsky "Winged-S" Lifesaving Rescue Award.

Certifying Tools for Test Reduction in Open Architecture

Valdis Berzins—Berzins is a professor of computer science at the Naval Postgraduate School. His research interests include software engineering, software architecture, reliability, computer-aided design, and software evolution. His work includes software testing, reuse, automatic software generation, architecture, requirements, prototyping, re-engineering, specification languages, and engineering databases. Berzins received BS, MS, EE, and PhD degrees from MIT and has been on the faculty at the University of Texas and the University of Minnesota. He has developed several specification languages, software tools for computer-aided software design, and fundamental theory of software merging. [berzins@nps.edu]

Abstract

In this paper, we describe a method for evaluating tools that can be used to guide decisions about how much retesting is needed and to check conditions under which testing of unmodified components can be reduced or avoided. The approach uses a combination of dependency analysis applied to source code and automated testing applied to executable component implementations. Dependability of such tools is a key concern in this context, which our ongoing research addresses. We also discuss other applications of software dependency analysis, such as risk-based testing, and discuss applications of dependency analysis to improve acquisition processes in the context of open architectures (OA).

The Navy's OA framework is intended to promote reuse, improve system flexibility, and reduce costs. In this paper, we apply open architecture principles to reduce testing effort and costs in cases where the requirements and code for a subsystem have not been changed, but the component will be used together with new or modified components that may include a new version of the operating system. This situation is common in the Navy due to technology advancement upgrades and accounts for a substantial fraction of the testing cost.

Applying traditional U.S. Navy weapon and combat system test and evaluation (T&E) practices, which currently include manual retesting after each system modification, to future OA systems will nullify many of the benefits that OA brings to the table, such as system scalability, rapid configuration changes, and effective component reuse. Combining (1) Naval Postgraduate School (NPS) research on dependency analysis focused on determining when it is safe not to retest a component with (2) automated software testing should enable these benefits and keep resource requirements at feasible levels.

Introduction

Open architecture (OA) promises cost savings and other benefits for the Navy. The weakest parts of this vision are the associated quality assurance (test & evaluation) processes and coordination among platform-oriented program offices (resource allocation). Our research has been aimed at providing conceptual and engineering support that can be leveraged to alleviate these weaknesses. In this paper, we report some recent results in these directions.

Traditional software testing techniques, like scenario-based integration testing, are commonly used for assessing the dependability of today's software systems. These techniques are strongly dependent on a particular system configuration and its platform. A major drawback is that when the system configuration or its platform changes, it is necessary to reconstruct the test cases and rerun them. Plugging in a new software component will lead to a completely different system and will likely invalidate previous test results, while changes to the cyber environment may reduce the effective coverage of the test scenarios previously used. This is the rationale for current test and evaluation policies that require complete regression testing of each new release.

The main challenge we perceive in test and evaluation is how to safely reduce duplication of effort when performing regression testing on new releases of systems with long lifetimes. The goal of regression testing is to verify that the behavior of software services that were not supposed to be changed from the previous release are not adversely affected by the new operating environment or interactions with new or modified code. Our overall approach to this challenge is

1. to utilize software slicing to determine when it is safe to reuse previous test results for components whose specifications and code remain unchanged from the previous release;
2. to utilize automated invariance testing to economically verify that modules whose code has changed but whose specifications have not changed relative to the previous release have not changed their behavior in the new release; and
3. to utilize operational profiles and associated analysis of probability distributions to determine which parts of the input space need additional test cases when a reusable component is deployed in a different environment (such as an additional platform) and to automatically generate the corresponding test data.

A road map for the overall approach can be found in Berzins, Rodriguez, and Wessman’s 2007 article “Putting Teeth into Open Architectures: Infrastructure for Reducing the Need for Retesting,” and details can be found in Berzins’s 2008 article, “Which Unchanged Components to Retest After a Technology Upgrade”; Berzins and Dailey’s 2009 article, “How to Check If It Is Safe Not to Retest a Component”; Berzins and Dailey’s 2010 article, “Improved Software Testing for Open Architecture”; and Berzins, Lim, and Kahia’s 2011 article, “Test Reduction in Open Architecture via Dependency Analysis.” In this paper, we provide additional material on new applications of software dependency analysis (software slicing) and on our preliminary work on assessing commercial software tools with respect to their ability to reliably carry out the necessary calculations to perform the analyses we propose.

Our approach to improving coordination between program offices that are involved with components shared across platforms or cross-cutting missions and capabilities that involve many platforms and systems is new in this paper and is discussed in the section titled Slicing for Risk-Based Acquisition.

Software Dependency Analysis

The main dependencies between software modules are associated with data flow and control flow, both of which can be mediated by networks. Additional dependencies are associated with exclusive access to shared resources, which are usually mediated by low-level locks. A dependency between two software modules means that the behavior of one module can be influenced by the behavior of the other.

Slicing for Safe Test Reduction

Program slicing (Weiser, 1984) is a type of dependency analysis that eliminates program statements irrelevant to a given slicing criterion. Slicing algorithms detect and follow dependencies of the kinds described above. A survey of known slicing algorithms can be found in Lim and Kahia (2011). Typical slicing criteria limit attention to the software behavior visible from a particular observation point, such as given output from the system or the result produced by a given software service. This means that a base program and its slice will produce the same results for the subset of interest (Gallagher & Harman, 1998). In particular, if a given software service has the same slice in the new release and the previous one, the results it produces will be the same in both releases, and it is only necessary to recheck any required timing constraints and resource constraints, such as memory and network bandwidth limitations. That can be done by focused stress testing, which takes much less effort than repeating a full set of regression tests for the given software service.

Slicing for Risk-Based Testing

Previous work on system risk assessment in the context of safety certification has combined the severity of potential failures with the estimated probabilities of occurrence to gauge risks associated with system operation, but this work is directly applicable at only the whole-system level. This is due to the fact that software-related operational hazards are mostly associated with the physical parts of the system, which are only indirectly affected by the software.

We are developing similar methods to determine how much testing and what other risk mitigation measures, if any, should be applied to each software component in an open architecture. However, the relationship between the individual embedded software components and the associated external effects is currently difficult to determine due to the size and complexity of practical software components and the lack of automated decision support. We are investigating how to apply software slicing and related dependency analysis techniques to solve this challenge. To our knowledge, this will be the first attempt to apply this new approach to address this system-of-systems challenge.

The intended principles of operation for the proposed software risk mitigation approach are as follows:

1. Perform a conventional whole-system operational risk analysis using approaches adapted from safety procedures certification, such as “MIL STD 882-D” (DoD, 2000). The result of this step is a list of potential mishap types associated with and ranked by their risk levels. We propose to extend the definition of mishaps in this context to include various aspects of mission failures that could be induced by system failures, in addition to the types of mishaps traditionally considered in a safety certification.
2. Perform a system level dependency trace to identify which subsystems affect each type of mishap listed in Step 1, and which software services affect each of those subsystems.
3. Perform a software dependency analysis using software slicing to identify which software modules affect each of the software services.
4. Using mishap list from Step 1 and the dependency relations from Steps 2–3, identify the set of potential mishaps that can be affected by each software module.
5. Associate the maximum-risk level of the set of mishaps identified in Step 4 with the corresponding software modules.
6. Use the risk level derived in Step 5 to determine the level of testing and possibly levels of additional risk mitigations to be associated with each software module.

The overall impact of adopting this approach would be more effective test and evaluation. If the worst-case risks associated with each component are known, then they can provide a principled and systematic basis for determining how much testing each component needs. This approach should be able to put the informal guideline to “test the most critical components the most thoroughly” on a sound quantitative basis, by providing an objective means for calculating how many test cases are needed for a given component, based on the associated risk levels. Some of the building blocks needed for this calculation can be found in Berzins's 2008 article, "Which Unchanged Components to Retest after a Technology Upgrade."

The testing approach described previously seeks to limit operational risks by running more test cases on the modules whose failure would lead to the most severe consequences. This is a statistical procedure aimed at reducing the likelihood of sampling errors—that is, reducing the likelihood that the finite set of test cases actually run passes successfully by pure chance, even though the failure rate of the component is actually unacceptably high. This kind of procedure can reduce failure rates but cannot guarantee complete absence of operational failures except in those rare cases where it is feasible to exhaustively test all possible combinations of system state and input values to the software component.

In cases where consequences of software failure are particularly severe, it is possible to modify the software design to provide additional structural risk mitigations. Some examples of these include

- Software safety interlocks: small, simple software modules whose only purpose is to isolate a safety-critical resource and to enforce safety constraints. This is typically done by limiting direct access to the critical resource to a small set of operations that can be effectively analyzed to show all possible sequences of operations satisfy the needed safety constraints.
- Redundant design: critical operations are realized by disparate, independently developed algorithms that are run in parallel. Results are compared and discrepancies are resolved by a voting procedure. This approach is effective if failures in the different algorithms are uncorrelated and the probability of any single algorithm failing on any possible input is less than 50%.
- Runtime monitoring: critical constraints are checked after each execution, and alternatives are used when the checks fail. This method is effective when there exist feasible mitigation strategies. For example, communication failures can sometimes be mitigated by retransmitting failed messages along different network paths, or processor failures can be mitigated by repeating the computation on different host hardware.

Slicing for Risk-Based Acquisition

Similar risk-based concerns arise on a larger scale in the acquisition of systems, particularly in desired future applications of open architectures in which software components are shared extensively across different platforms. In this context, risk severity would measure the impact of possible failures of system capabilities or supporting software services on different types of missions. The risk analysis process described in the previous section can be extended to support such enterprise-wide cost/benefit analysis.

The resulting analysis would support system-wide and enterprise-wide planning in the overall acquisition process by providing objective measures of the benefits/risks associated with implementing particular capabilities and services in a given system release or deciding to delay those capabilities and services to a later release. Having such measures could alleviate the tendency to reach globally sub-optimal acquisition planning due to independent local optimization by each program office, which is typically responsible for a single platform. Cross-cutting concerns that affect the interaction of multiple platforms, such as those sharing a given component, do not have an advocate with authority to carry out enterprise-wide optimization in the current process. Lacking centralized authority, visible measures of the enterprise-wide tradeoffs could provide an objective basis for negotiation and cooperation between program offices impacted by cross-cutting concerns, given appropriate leadership and encouragement. Such cooperation would be very difficult without accepted objective measures of the impact of alternative decisions, because program offices compete for resources and each has its own set of local priorities.

We suggest extending slicing and dependency analysis from code level to the software and system architecture levels to support the process described above. If developed to the point where the enterprise-wide impacts of deployment decisions related to individual components can be objectively measured, this can improve consistency of priorities across the enterprise. In the best case, it could reduce the waste associated with situations where a capability has been implemented on a given platform but is not accessible from the other platforms that need that capability to successfully cooperate to perform a larger mission. For this purpose, it would not be necessary to provide quantitative estimates of the risk/benefits of various missions; priorities determined by rank-ordering the importance of various missions would be sufficient.

Robust composite systems can be composed of components in a disciplined manner, according to carefully designed software architecture. A *software architecture* consists of

1. a set of components,
2. an interconnection pattern for the components, and
3. a set of constraints on the components and connections.

The constraints typically express various kinds of requirements associated with the entire system, as well as with the components and connections of the architecture (*Proceedings of the 4th Monterey Workshop*, 1995; Luqi, Qiao, & Zhang, 2002). We propose that the development and analysis of robust architectures be combined with methods for determining the priority order for implementing and deploying a variety of system refinements and extensions that can be supported by the architecture. These methods should provide guidance for the effective allocation of development resources and schedules in the context of phased acquisition planning for a series of system releases.

The priority of a mission should be inherited by all of the capabilities and subsystems on which the mission is dependent. In the simplest case, a component supporting many different missions would get the priority of the most important mission it supports. This simple case assumes that independently meaningful priorities can be assigned to individual missions, which is not always the case, as pointed out by a common criticism of the well-known analytic hierarchy process (AHP) for assigning priorities based on pairwise comparisons (Saaty, 2008). For example, it will be difficult for a soldier to say which is more important: having a weapon or having ammunition. The reason is that neither one in isolation has much value—the soldier needs to have both for them to make a difference. In cases where bundles of missions have this kind of priority interaction, the analysis model can represent a bundle of synergistic missions as an indivisible unit that can be assigned a meaningful priority. Each individual mission that belongs to at least one such mission bundle can inherit the highest priority of all the mission bundles it belongs to.

The proposed priority inheritance pattern is illustrated by the small example in Tables 1 and 2.

Table 1.

Mission Group Priorities		
Mission Bundle	Priority	Members
Bundle 1	High	M1, M2
Bundle 2	Medium	M1, M3

Table 2.

Inherited Priorities for Individual Missions	
M1	High
M2	High
M3	Medium

Note that mission M1 belongs to two different mission bundles, and that its priority is the highest of the priorities of the two bundles it belongs to. The priority assignment illustrated would imply that the available choices are (1) to fund Bundle 1 alone, which would support missions M1 and M2; (2) to fund both Bundle 1 and Bundle 2, which would support missions M1, M2, and M3; or (3) to fund neither of them. Similar processes can be applied at the level of system components or system services supporting each mission.

Although assigning priorities is always a contentious issue, our hypothesis is that it is easier to get agreement on the relative priorities of different types of missions than it is to get agreement on priorities of particular system components. If a cross-cutting mission is important enough to be resourced, then we propose that it should get all of the components needed to make it operational, regardless of which program offices are responsible for those components. It does not make sense to deploy incomplete parts of capabilities needed to support an important mission any more than it does to issue a weapon to a soldier without ammunition or ammunition without a weapon. If this proposition is accepted, architecture-level dependency analysis relative to sets of missions will be needed to translate priorities of missions and mission groups into corresponding priorities of system components and services. Suitable extensions of slicing technology and associated tool support can help carry out that analysis on a large scale.

Game-Theoretic Risk Analysis

The section titled Slicing for Risk-Based Testing describes a conventional statistical approach to risk analysis and mitigation. Such an approach is sensible when the dominant root cause of the failures we are trying to prevent is random failure of physical components, or random variations in operating conditions and input data that could exercise fixed but unknown software faults that could in turn trigger actual failures or mishaps.

Such a model may not be adequate for assessing risks due to possible cyber-attacks or other forms of deliberate enemy action. In this case, stationary probability distributions do not provide an accurate view of the expected frequency of mishaps, unless we assume incompetent adversaries. Given the military’s concern with nation-state actors, that is not a safe assumption to make.

Saddle point solutions from game theory may provide a more accurate representation of risk situations associated with deliberate attacks. A saddle point solution minimizes the worst-case damage that could be inflicted by any action the adversary could feasibly perform. Some examples of this type of risk analysis in the context of security aspects of cyber-physical systems can be found in Andreasson et al. (2011) and Zhu and Basar (2011). We are investigating the hypothesis that this approach can be combined with slicing to support both risk-based testing and risk-based acquisition.

Slicing Tools

Our preliminary analysis has identified Indus's static slicing tool for Java programs as the most promising candidate for detailed assessment, as well as some of the criteria for evaluating such tools and a few test cases derived from the criteria (Berzins et al., 2011). Other tools were identified but not selected for detailed evaluation due to licensing issues, concerns about tool stability, lack of support, and the relevance of the target programming language supported by the tool (Lim & Kahia, 2011).

Tool Certification Approach

Evaluation Criteria

We have proposed a set of evaluation criteria for slicing tools (Berzins et al., 2011; Lim & Kahia, 2011). These criteria can be summarized as follows

- The tool must generate sound slices: all statements that can affect software behavior at the specified observation point must be included.
- The tool must operate on at least one programming language of concern in open architectures. Those include Java, C/C++, and Ada.
- The tool must have adequate documentation and support.
- Installation procedures and tool operation must work as advertised.
- The tool must support output and/or comparison of computed slices.

The first criterion is necessary for safe reduction of testing. It implies that the tool must handle all language features used in realistic applications, and must properly detect the associated dependencies. This is a potential issue because many of the slicing algorithms developed by academic research were published and described in terms of simplified models of programming languages. Slicing algorithms capable of properly analyzing some programming language features in common practical use have only recently been developed and published. Language features of this kind include the following:

- pointers and objects,
- parallel threads of control,
- exceptions,
- nondeterministic selection,
- locks and synchronizations, and
- external dependencies: libraries, databases, etc.

Tool capabilities to handle these software features should be explicitly assessed via test cases because there is a plausible risk that the relevant recent research results may not have been completely or correctly implemented in available commercial tools.

In addition to the sharp requirements identified above, slicing tools have a *preference metric*: the smaller the slice, the more effective the tool will be in reducing redundant regression testing. It is not possible to get a perfect solution to this problem because finding exact minimum size slices is known to be algorithmically unsolvable (Weiser, 1984). The entire program is always a valid slice but has no discrimination power and is useless for test effort reduction. Practical tools seek to find "small" slices that are not guaranteed to have globally minimal size. Therefore, tools should be compared for relative discrimination power using the above preference metric.

Assessment Approach

We will conduct a two-phase assessment. The first phase will consist of running a hand-crafted set of test cases aimed at checking the criteria, given in the previous section, with particular attention to language features and issues that have been noted as potentially problematic in the literature. A preliminary set of such test cases can be found in Lim and Kahia (2011).

The second phase will consist of a larger case study. We are currently seeking a suitable open source software system suitable for evaluating Indus's Kaveri slicing tool. The key features we seek are

- a design with multiple modules and subsystems,
- access to multiple releases of the source code,
- access to specifications or descriptions of the versions that at a minimum identify which services were intended to be modified in each release, and
- access to test results and problem reports for each version.

The proposed assessment will use the slicing analysis to determine which modules were safe not to retest and then compare the conclusions with actual test results and problem reports from the field. The plan is to analyze versions that are old enough to have post-mortem information about field experiences available.

Conclusion

The work reported here is a step towards an affordable approach to providing robust adaptable software systems in the context of open architectures.

Robust adaptive software design requires substantial architectural support. Sound architectural models for cyber systems with adaptable software components and associated quality assurance methods collectively gain the ability to replace bits of systems while maintaining system dependability (Hinchey & Vasev, 2010). System design with recorded rationale can make testing of adaptable cyber systems possible.

This requires a shift from scenario-based testing to architecture-based quality assurance (Luqi, Zhang, Berzins, & Qiao, 2004; Qiao & Luqi, 2004; Luqi, 2006), along with a shift from code-based adaptation to architecture based adaptation (Oreizy et al., 1999; Garlan, Cheng, Huang, Schmerl, & Steenkiste, 2004; Calinescu, 2009). A good cyber system architecture would have associated dependability properties that express stable system requirements, requirements on the subsystems, and a sound software evolution model capturing the design rationale (Rajkumar, Lee, Sha, & Stankovic, 2010). The architecture itself would provide some degree of dependability guarantees, regardless of specific configuration. Testing and analysis would be applied to a suitably specified architectural model in addition to the system implementation. Software slicing is one aspect of this analysis.

Software slicing is a realistic approach to safely reducing regression testing in adaptable systems. Practical application requires dependable tools for software slicing. Ongoing research is conducting an evaluation of such tools. Preliminary results and additional potential applications of such tools are presented in this paper.

References

Andreasson, M., Amin, S., Schwartz, G., Johansson, K., Sandberg H., & Sastry, S. (2011, April). Correlated failures of power systems: Analysis of the Nordic grid. In *Proceedings of Workshop on the Foundations of Dependable and Secure Cyber-Physical Systems* (pp. 9–17).

Berzins, V. (2008, May). Which unchanged components to retest after a technology upgrade. In *Proceedings of the Fourth Annual Acquisition Research Symposium: –Creating Synergy for Informed Change* (pp. 142–153). Monterey, CA: Naval Postgraduate School.

Berzins, V., & Dailey, P. (2009, May). How to check if it is safe not to retest a component. In *Proceedings of the Sixth Annual Acquisition Research Symposium: Defense Acquisition in Transition* (pp. 189–200). Monterey, CA: Naval Postgraduate School.

Berzins, V., & Dailey, P. (2010, May). Improved software testing for open architecture. In *Proceedings of the Seventh Annual Acquisition Research Symposium: Creating Synergy for Informed Change* (pp. 385–398). Monterey, CA: Naval Postgraduate School.

Berzins, V., Lim, P., & Kahia, M. B. (2011, May). Test reduction in open architecture via dependency analysis. In *Proceedings of the Eighth Annual Acquisition Research Symposium* (pp. 333–344). Monterey, CA: Naval Postgraduate School.

Berzins, V., Rodriguez, M., & Wessman, M. (2007, May). Putting teeth into open architectures: Infrastructure for reducing the need for retesting. In *Proceedings of the Fourth Annual Acquisition Research Symposium: –Creating Synergy for Informed Change* (pp. 285–312). Monterey, CA: Naval Postgraduate School.

Calinescu, R. (2009). Reconfigurable service-oriented architecture for autonomic computing. *International Journal on Advances in Intelligent Systems, 2*(1), 38–57.

DoD. (2000, February 10). MIL STD 882D - Standard practice for system safety. Retrieved from http://www.everyspec.com

Gallagher, K., & Harman, M. (1998). Program slicing [Special issue]. *Information and Software Technology, 40*(11/12), 1–10.

Garlan, D., Cheng, S., Huang, A., Schmerl, B., & Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer, 37*(10), 46–54.

Hinchey, M., & Vasev, E. (2010). Software verification of autonomic systems developed with ASSL. In *Monterey Workshop on Modeling, Development, and Verification of Adaptive Systems* (pp. 7–15).

Lim, P., & Kahia, M. B. (2011, June). *Suitability of commercial slicing tools for safe reduction of the testing effort* (Master’s thesis, Naval Postgraduate School). Monterey, CA: Naval Postgraduate School.

Luqi. (2006, May 6). Transforming documents to evolve high-confidence systems. In *Proceedings of Workshop on Advances in Computer Science and Engineering* (pp. 71–72).

Luqi, Qiao, Y., & Zhang, L. (2002, October). Computational model for high-confidence embedded system development. In *Monterey Workshop: Radical Innovation of Software and System Engineering in the Future* (pp. 265–303).

Luqi, Zhang, L., Berzins, V., & Qiao, Y. (2004). Documentation driven development for complex real-time systems. *IEEE Transactions on Software Engineering, 30*(12), 936–952.

Oreizy, P., Gorlick, M., Taylor, R., Heimbigner, D., Johnson, G., Medvidovic, . . . Wolf, A. (1999). An architecture-based approach to self-adaptive software. *Intelligent Systems and their Applications, IEEE, 14*(3), 54–62.

Proceedings of the 4th Monterey workshop on specification based software architectures. (1995). Monterey, CA.

Qiao, Y., & Luqi. (2004). Admission control for dynamic software reconfiguration in systems of embedded Systems. In *Proceedings of the 2004 International Conference on Embedded Systems and Applications.*

Rajkumar, R., Lee, I., Sha, L., & Stankovic, J. (2010). Cyber-physical systems: The next computing revolution. In *Proceedings of the 47th Design Automation Conference* (pp. 731–736).

Saaty, T. (2008). Relative measurement and its generalization in decision making: Why pairwise comparisons are central in mathematics for the measurement of intangible factors: The analytic hierarchy/network process. *RACSAM (Review of the Royal Spanish Academy of Sciences, Series A, Mathematics), 102*(2), 251–318.

Weiser, M. (1984). Program slicing. *IEEE Transactions on Software Engineering, 10*(4), 352–357.

Zhu, Q., & Basar, T. (2011). Towards a unifying security framework for cyber-physical systems. In *Proceedings of Workshop on Foundations of Dependable and Secure Cyber-Physical Systems* (pp. 47–50).

NPS
PRAESTANTIA PER SCIENTIAM
1909
Acquisition Research Program:
Creating Synergy for Informed Change

Certifying Tools for Test Reduction in Open Architecture

Valdis Berzins

Naval Postgraduate School

U.S. Navy Open Architecture

- **A multi-faceted strategy for developing joint interoperable systems that adapt and exploit open system design principles and architectures**

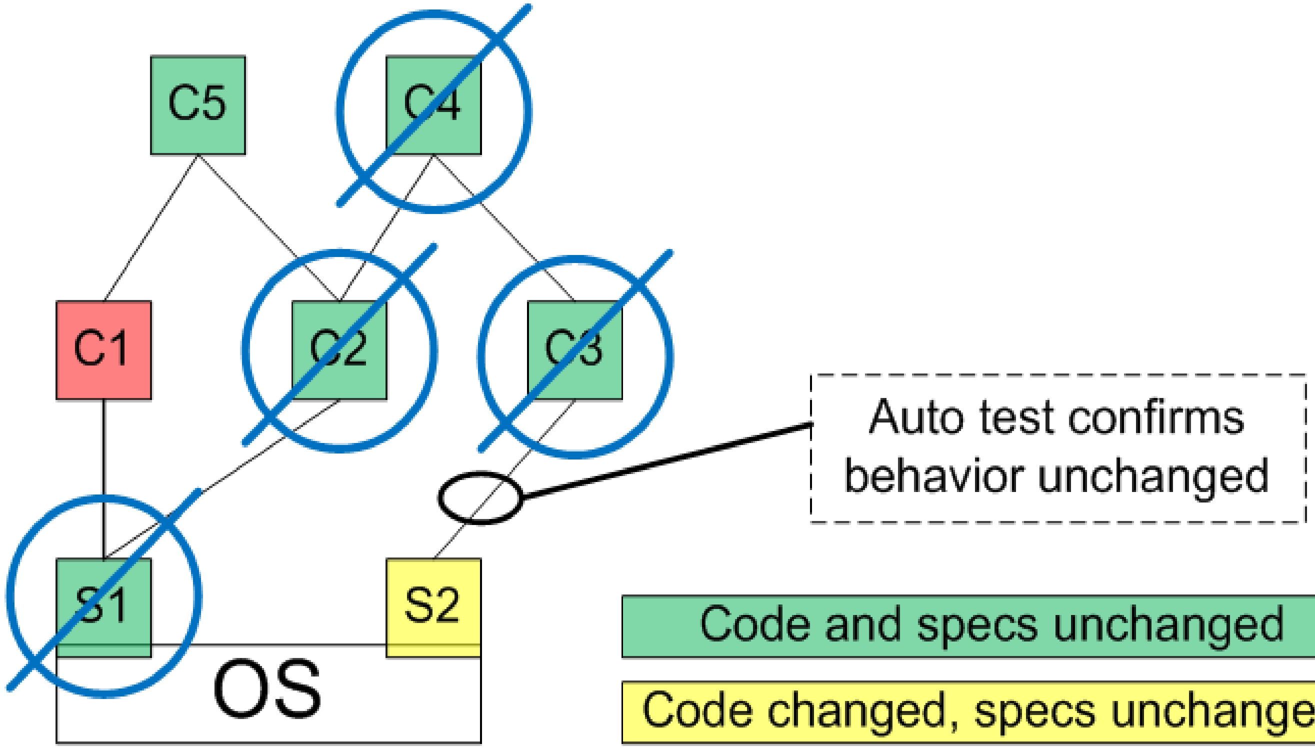
- **OA Principles, processes, and best practices:**
  - Provide more opportunities for completion and innovation
  - **Rapidly field affordable, interoperable systems**
  - **Minimize total ownership cost**
  - Maximize total system performance
  - Field systems that are easily developed and upgradable
  - Achieve component software reuse

Problem and Proposed Solution

- **Traditional U.S. Navy Software T&E practices will limit many benefits of OA**
  - It is virtually impossible to field frequent and rapid configuration changes with current approaches
- **New Testing Technologies, Processes & Policies are Needed**
  - Safely Reduce Testing Required (2007-2012)
  - Make testing more effective
    - Risk-based testing (2012), safe test result reuse (Berzins, 2009)
  - Transition from Manual Testing to Profile-Based Automated Statistical Testing (Berzins, 2010)
  - Dependency-based acquisition (2012)

Test Avoidance Approach

= No retest due to slicing and invariance testing



🟩 Code and specs unchanged

🟨 Code changed, specs unchanged

🟥 Code and specs changed

Program Slicing

- Program slicing is a kind of automated dependency analysis
  - Same slice implies same behavior
  - Can be computed for large programs
  - Depends on the source code, language specific
  - Some tools exist, but are not in widespread use
- Slicing tools must handle the full programming language correctly to support safe reduction of testing.

Test Reduction Process

- Check that the slice of each service is the same in both versions (automated)
- Check that the requirements and workload of each service are the same in both versions
- Must recheck timing and resource constraints
- Must certify absence of memory corrupting bugs
  - Popular tools exist: Valgrind, Insure++, Coverity, etc.
- Must ensure absence of runtime code modifications due to cyber attacks or physical faults
  - Cannot be detected by testing because modifications are not present in test loads
  - Need runtime certification
    - Can be done using cryptographic signatures (Berzins, 2009)

The Current Problem

To Evaluate the Suitability of

COTS Slicing Tools

for Supporting Safe Test Reduction

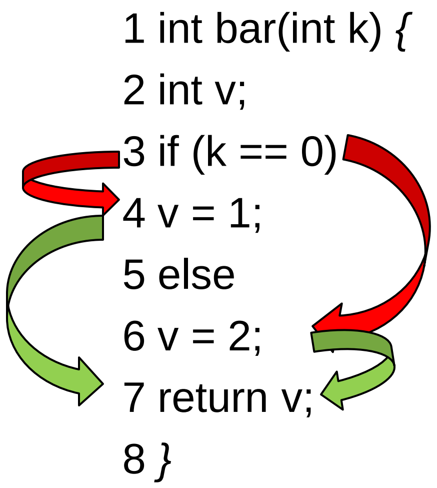
Current Research Objectives

1. To conduct experimental assessments and compare the suitability of the available COTS program slicing tools for safe reduction of testing effort.
2. To identify the most adequate slicing tools among the evaluated ones.
3. To determine the suitability of available COTS program slicing tools for practical SW test reduction.
4. To explore additional benefits of dependency analysis

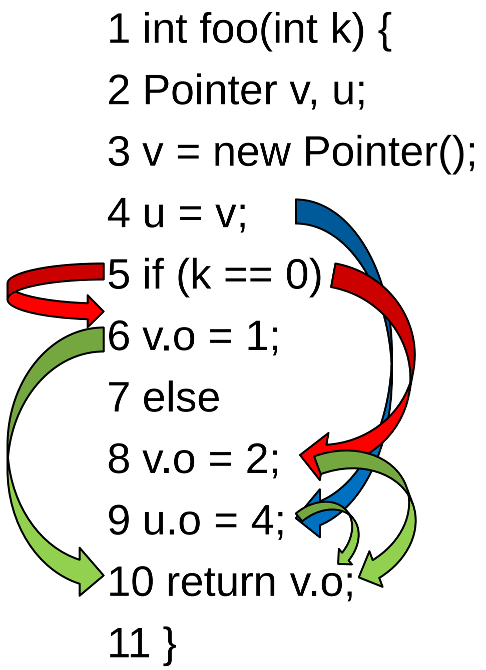
Requirements for Slicing Tools

1. Must satisfy the behavior invariance property:
   - If the original program terminates cleanly, the slices must terminate cleanly and produce the same result as the original program for all observable values specified by the slicing criterion.
2. Must support comparison or output of computed slices
3. Must support modeling of external dependencies

Examples of Dependencies



```
1 int bar(int k) {
2 int v;
3 if (k == 0)
4 v = 1;
5 else
6 v = 2;
7 return v;
8 }
```



```
1 int foo(int k) {
2 Pointer v, u;
3 v = new Pointer();
4 u = v;
5 if (k == 0)
6 v.o = 1;
7 else
8 v.o = 2;
9 u.o = 4;
10 return v.o;
11 }
```

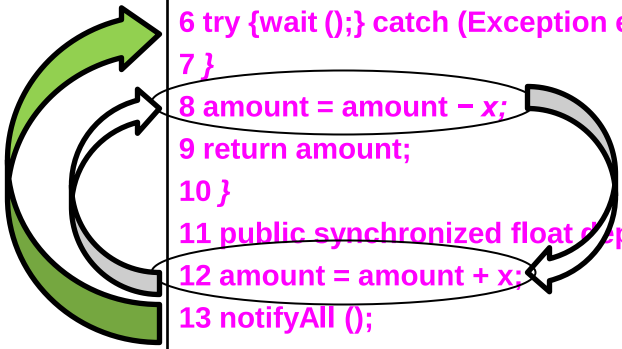
Legend

🟥 Control Dependency

🟩 Data Dependency

🟦 Pointer Aliasing Dependency

Examples of Parallel Dependencies



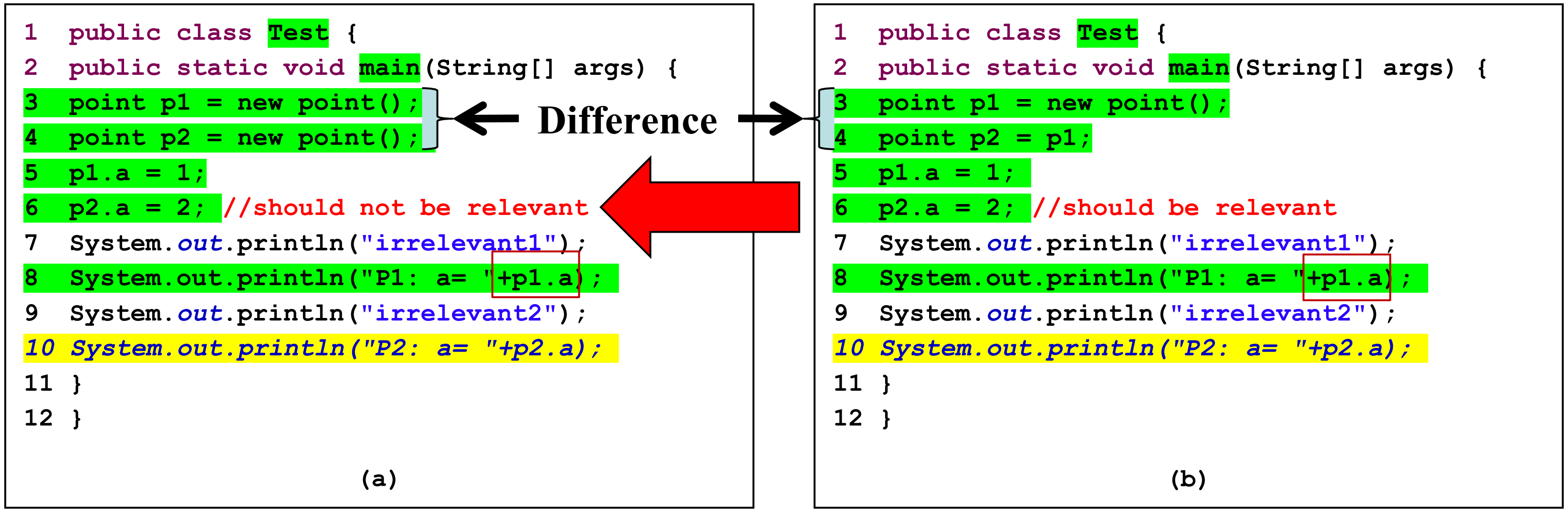
```
class Account {
private float amount = 0;

public synchronized float withdraw(float x) {
while (amount − x < 0) {
try {wait ();} catch (Exception e) { }
}
amount = amount − x;
return amount;
}
public synchronized float deposit(float x) {
amount = amount + x;
notifyAll ();
return amount;
}
}

class Worker implements Runnable {
private Account save;
private float amount;
public Worker(Account account, float a) {
save = account;
amount = a;}
public void run() {
save.deposit(amount);
}
}
```

```

class Spouse implements Runnable {
private Account save;
private float amount;
public Spouse(Account account, float a) {
save = account;
amount = a;}
public void run() {
save.withdraw(amount);
(new Account()).deposit(10);
}
}
```

```

class Home {
public static void main(String[] s) {
Account savings = new Account();
Runnable worker = new Worker(savings, 90);
Runnable spouse = new Spouse(savings, 10);
new Thread(worker).start();
new Thread(spouse).start();
}
}
```

Slicing Example

Resolution of slices computed by Kaveri



```
1  public class Test {
2  public static void main(String[] args) {
3  point p1 = new point();
4  point p2 = new point();
5  p1.a = 1;
6  p2.a = 2; //should not be relevant
7  System.out.println("irrelevant1");
8  System.out.println("P1: a= "+p1.a);
9  System.out.println("irrelevant2");
10 System.out.println("P2: a= "+p2.a);
11 }
12 }
```

(a)

```
1  public class Test {
2  public static void main(String[] args) {
3  point p1 = new point();
4  point p2 = p1;
5  p1.a = 1;
6  p2.a = 2; //should be relevant
7  System.out.println("irrelevant1");
8  System.out.println("P1: a= "+p1.a);
9  System.out.println("irrelevant2");
10 System.out.println("P2: a= "+p2.a);
11 }
12 }
```

(b)

Using slicing criterion {8, p1.a} for both (a) and (b)

Legend:

🟨 Partially Relevant Slice

🟩 100% Relevant Slice

Project Status

- Experimental assessment is in progress and not yet complete.
  - The team is currently instrumenting the tools and developing additional test cases.
- Developed the initial framework for two additional uses of dependency analysis:
  - Risk based testing
  - Risk based acquisition

Risk Based Testing

1. Whole-system operational risk analysis identify potential mishaps / mission failures
2. Identify which software service failures would lead to identified mishaps
3. Use slicing to identify which software modules affect the critical services
4. Associate maximum risk level of affected services with each software module
5. Set number of test cases using risk level

Current Policy for Mishap Risk Assessment

FREQUENCY OF OCCURRENCE	MISHAP SEVERITY CATEGORIES			
	1 CATASTROPHIC	2 CRITICAL	3 MARGINAL	4 NEGLIGIBLE
A – FREQUENT P ≥ 10%	1A	2A	3A	4A
B – PROBABLE 10% > P ≥ 1%	1B	2B	3B	4B
C – OCCASIONAL 1% > P ≥ 0.1%	1C	2C	3C	4C
D – REMOTE .1% > P ≥ 0.0001%	1D	2D	3D	4D
E – IMPROBABLE 0.0001% > P	1E	2E	3E	4E

Cells:	Risk Level & Acceptance Authority:
1A, 1B, 1C, 2A, 2B:	**HIGH** – ASN (RDA)
1D, 2C, 3A, 3B:	**SERIOUS** - PEO-IWS
1E, 2D, 2E, 3C, 3D, 3E, 4A, 4B:	**MEDIUM** –PEO-IWS 3
4C, 4D, 4E:	**LOW** – PEO-IWS 3

P: Probability of occurrence in the lifetime of an individual system, ranges taken from MIL_STD-882D

Risk Based Acquisition

1. Identify missions and scenarios that systems must support
2. Assign priorities to missions / scenarios based on impact of success or failure
3. Use dependency analysis to identify which system components affect mission success
4. Associate maximum priority of affected missions / scenarios with each component
5. Allocate funding per priority level, regardless of which program offices are responsible.

Example

Mission Group Priorities		
Mission Bundle	Priority	Members
Bundle 1	High	M1, M2
Bundle 2	Medium	M1, M3

Inherited Priorities for Individual Missions	
M1	High
M2	High
M3	Medium

Priorities of different bundles must be different

Assumptions

1. It is less contentious to prioritize missions and scenarios than system components
2. In the absence of cross-cutting budget authority, a principled basis for cross-cutting allocation is needed to reach agreement.
3. As more components are shared across platforms, such issues will gain importance.

Conclusion

- For systems with long lifetimes, regression testing is a major cost component in each new release, including periodic technology upgrades.
- Program Slicing has the potential to reduce the time and cost of the regression testing that is necessary to ensure the safety and effectiveness of each new release.
- Preliminary evaluation criteria for slicing tools in the context of their ability to achieve safe reduction of regression testing have been developed.

Thank you